REMARKS

I. Status of the Claims:

Claims 1-11 and 13-30 are currently pending. By this Amendment, claims 15, 18 and 30 have been amended. No new matter has been introduced by this Amendment. Upon entry of this Amendment, claims 1-11 and 13-30 would be pending.

II. Specification:

The specification was objected to on matters of formality, e.g., lacking particular section headings. The specification has been amended in accordance with the Examiner's suggestions.

III. Rejection Under 35 U.S.C. § 112:

Claims 15, 18 and 30 are rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter.

The claims have been amended to address the Examiner's concerns.

IV. Rejection Under 35 U.S.C. § 102 & § 103:

Claims 1-5, 13-18 and 26-30 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Kazi et al. Claims 6-11 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Kazi et al. in view of Miller et al. The Applicants respectfully traverse the rejection of the claims.

A. Claims 1 to 5, 12 to 18, 26 to 30:

Both independent claims 1 and 13 are directed to methods involving translating the program bytecode into machine independent virtual processor (VP) code which uses an instruction set of a virtual processor; transmitting the virtual processor code from a server to a client device; and translating the virtual processor code into native code which uses an instruction set of a physical processor of the client device. The bytecode is stack-based, and the virtual processor code is register-based.

In other words, a register based intermediate representation which in the art has only been used internally is distributed to client computers. As noted below and in the prior reply, this would for example involve adaptation of the intermediate representation.

Further, for the Examiner's reference, in the prior art, the final step of compilation to native code occurs at one and the same place as the translation to the intermediate language. Thus, for each processor type, a different native code needs to distributed. The claimed arrangements however involve for example distribution of one (register based) intermediate representation to clients with different processors. Because of this, one intermediate representation rather than several native ones need to be maintained at the server. Because the claimed intermediate representation is register based (rather than stack based as JBC), it is closer to the native code and hence final compilation is more efficient as compared to e.g. distribution of JBC. The resulting system which in effect remotes the stages of compilation to different machines is overall more efficient due to the combination of storage of only one intermediate representation at a server and efficient compilation to native code at a client device.

The cited references simply do not disclose or suggest the claimed arrangements, and provide nothing in the way of motivation for modifying the cited references in the manner suggested by the Examiner absent impermissible hindsight.

First, the Examiner appears to equate the claimed VP code with Java Bytecode (JBC), in spite of the clear limitation that VP code is register-based (whereas JBC is stack based). VP code is however different from JBC as relied upon by the Examiner. The Examiner's position also ignores the clear teaching in all of the prior art that any "register-based" (i.e., closer to processor architecture/native code) intermediate code which may be used in some of the cited prior art is an "internal" intermediate representation internal to the compilers in question. Taking such an internal intermediate representation and turning it into something which is transmitted over a network is nowhere disclosed or even suggested in the cited references. Further, as previously argued in the prior reply, this is not enabled by the cited references.

Second, the Applicants respectfully submit that taking a representation (register based VP code) the likes of which have previously only been used internally in compilers and transmitting it over a network is not obvious. The fact that other, stack based intermediate representations such as JBC are transmitted over a network has no bearing on this. The distribution of a register based intermediate representation is not disclosed or suggested anywhere in the cited references.

Finally, it is further respectfully submitted that a prima facie case of obviousness has not been established and, in any event, has been rebutted in the prior reply of which points three and five have not been addressed at all by the Examiner.

Specifically, as noted in the third point, obviousness cannot be established with art that teaches away. In evaluating obviousness it is not appropriate to selectively consider part of a reference while ignoring other parts that teach away from the invention. As stated in MPEP § 2141.02, a prior art reference must be considered in its entirety, i.e. as a whole, including

portions that would lead away from the claimed invention. It is stated throughout Kazi that the intermediate representations are *"internal"* to the compiler. They are thus clearly **not** transmitted via the networks. For example, see page 14, second full paragraph starting *"caffeine..."*, third line and page 15, first full paragraph starting *"the native executable translation..."*, line 8. In as far as Figure 2 relates to JIT compilers, these compile bite code at run time i.e. at the client device if such a device were mentioned at all (see page 9, first paragraph of heading 3.2.2, lines 2 to 4). If at all, it is therefore clearly the byte code which should be transmitted according to Kazi and not any intermediate representation.

Further, as noted in the fifth point, even if one of ordinary skill in the art, starting from Kazi, tries to implement a system in which an intermediate representation is sent over a network and then compiled at the client device, the teaching of Kazi would not actually enable him to do so for the following reasons.

The intermediate codes referred to in Kazi cannot simply be transmitted over the network because they are internal products of the compiler (see above). Although Kazi is a review of many different state of the art compilers, it fails to show how these internal products could be changed to send them over a network, nor indeed is there any indication that this could or has been done. The Java bytecode class file in Figure 2 includes in addition to the byte code instructions for the methods of the class data which allows methods of the class to reference each other and to be referenced by methods of other classes (see page 12, *"constant pool"*, *"methods ??"* and information relating the class to other classes). This information of the class files is used to fix the references in the methods of the class such that they can be resolved when the resulting native code is executed. This happens either at run time in a java virtual machine or on compilation into native code in direct compilers.

If one were to take the internal representations referred to in Kazi and transmit them over a network for compilation into native code, the resulting system would not work because the claimed device would have no information on how to resolve the references in the intermediate language. There is no suggestion in Kazi of how this could be achieved and therefore, even if one of ordinary skill in the art would attempt to modify the system disclosed in Kazi as suggested by the Examiner, such a person would end up with a system which does not work because there is no disclosure in Kazi how the intermediate representations could be turned into native code once they have been transmitted to the client device.

In view of the foregoing, claims 1 and 13 and their dependent claims and the other rejected claims are believed to be distinguishable over the cited references, individually or in combination. Reconsideration and withdrawal of the rejection of the claims are respectfully requested.

B. Dependent Claims 6 to 10:

Claims 6-11 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Kazi et al. in view of Miller et al. The Applicant respectfully traverses this rejection for the reasons set forth below.

The Examiner asserts that the Applicants have failed to point out where the novelty in the claims are pursuant to MPEP 714.04. The Applicants respectfully disagree and submit that the Examiner has completely ignored the specific distinctive language of the claims and its related arguments as properly set forth in the prior response. As such, these arguments are respectfully resubmitted for consideration.

Specifically, as previously argued, these claims relate to *"fixing up"* (e.g., fixup of) the byte code when it is translated into virtual processor code, which means that instructions are provided which allow the references to a field within the class to be located by the native code (see claim 5) or the reference to another class or field or method in another class to be resolved (see claim 10). See also page 11 line 26, page 17 line 4 for a detailed description of these topics. By contrast, the disclosure of Miller, and in particular the portions referred to by the Examiner, relate to a programming language which has size-indefinite variables and its compilation into native code, including generating a size-definite variable corresponding to the size-indefinite variable according to machine specific criteria. The portion cited by the Examiner relate either to turning an intermediate language into native code, generally (for example column 1 lines 15 to 20, column 6 lines 1 to 10), or the determination of the size of the variable in the native code (column 7 line 20 to 30 and 37 to 46, column 6 lines 30 to 36, for example). *There is no disclosure of resolving references within or between classes, let alone any reference to "fixing up" as described above.* Accordingly, Miller cannot make up for the shortfall in Kazi and even a combination of these two references fails to teach or suggest all the elements of the respective claims.

Further, it is noted that any combination (which would still fail to disclose all features of the respective claims) of Kazi and Miller would only be possible with the inadmissible use of impermissible hindsight because Kazi relates to the efficient compilation of Java programs and Miller relates to indefinite size intermediate languages. The latter are not mentioned or referenced in Kazi or not knowledge available to one of ordinary skill in the art, and there would be no reason for the skilled person to combine these two references.

In view of the foregoing, claims 6-10 are further distinguishable over the cited references, individually or in combination. Reconsideration and withdrawal of the rejection of the claims are respectfully requested.

V. Rejection Under 35 U.S.C. § 102:

Claims 19-25 are rejected under 35 U.S.C. § 102(e) as being anticipated by Miller et al. (US 6,389,590). Claims 19-25 are rejected under 35 U.S.C. § 102(e) as being anticipated by Kazi et al. ("Techniques for Obtaining High Performance in Java Program," 7-1999). Claims 19-25 are rejected under 35 U.S.C. § 102(e) as being anticipated by Koizumi et al. (US 5,586,323). The Applicants respectfully traverse the rejection of these claims.

Claim 19 is directed to a distributed computer system comprising a server including a store for storing virtual processor code, said code being a machine-independent representation of the bytecode of an object oriented computer program using an instruction set of a virtual processor, and a plurality of remote client devices in communication with the server, each client device including a client processor, a native translator arranged to translate the virtual processor code into native code which uses the instruction set of the respective client processor, and a native code store; the system including transmission means for transmitting the virtual processor code from the server to the client devices. The bytecode is stack-based, and the virtual processor code is register-based.

Claim 19 specifies that the byte code is stack-based and the virtual processor code is register-based and that the virtual processor code is a representation of the byte code of an object or a related computer program. Claim 19 thus corresponds to independent claim 1.

The Examiner asserts that the Applicants have failed to point out where the novelty in the claims are pursuant to MPEP 714.04. The Applicants respectfully disagree and submit that the Examiner has completely ignored the specific distinctive language of the claims and its related arguments as properly set forth in the prior response. Throughout the Office Action, the Examiner has not addressed the claims with respect to a "virtual processor code" that is register-based and is a representation of the byte code of an object or a related computer programs in any manner, and its context with the other elements of the claim language. The cited references are clearly silent as to any sending of a register-based intermediate code over a network. Accordingly, the Applicants respectfully submit that the claims are distinguishable over the cited references and resubmit the prior arguments for proper consideration.

For example, in these rejections, the Examiner seems to equate virtual processor code with byte code. This is clearly not consistent with the present wording of claim 19 which explicitly states that it is bytecode which is transformed into virtual processor code and further makes it clear that bytecode is stack-based whereas virtual processor code is register-based. For this and other reasons set forth below, claim 19 and its dependent claims are not anticipated and are distinguishable over each of the cited references.

Specifically, turning to Kazi, with respect to independent claim 1, the Examiner acknowledges that Kazi does not disclose step (b) of claim 1, that is transmitting the virtual processor code from a server to a client device. In claim 19, the virtual processor code is stored on the server and is translated into native code by the client. Therefore, the virtual processor code must necessarily be transmitted from the server to the client. However, as acknowledged by the Examiner in relation to claim 1, this feature is not disclosed in Kazi. Thus, claim 19 is not

anticipated by Kazi and likewise would not be rendered obvious by the cited references for similar reasons discussed above with reference to claim 1.

Claim 19 also stands rejected as being anticipated by Miller (point 6 of the Office Action). However, there is no disclosure of a virtual processor code which is register-based in Miller. As set out on pages 9 and 10 of the Applicant's last response, the virtual processor code is designed to be closely related to native code for native processors. For example, it is register-based rather than stack-based, as claimed. Miller teaches against this very idea because Miller's intermediate language is designed to handle indefinite sized variables, which requires more rather than less processing for conversion into the native code (it is this extra processing to which Miller relates). Therefore, Miller actually teaches against the present invention as defined in claim 19.

Finally, claim 19 stands rejected as being anticipated by Koizumi *et al.* However, Koizumi *et al* relates to the translation of source code into an intermediate representation, wherein the claimed invention specifically relates to the translation of bytecode (itself an intermediate representation, see above) into a further intermediate representation, that is virtual processor code. Accordingly, claim 19 is not anticipated by Koizumi.

Furthermore, there is nothing in Koizumi which would motivate one of ordinary skill in the art to translate the source code into a further intermediate representation (bytecode) before translating it into the intermediate representation used in Koizumi. Byte code is already a machine independent intermediate representation which can be executed on various platforms using a Java Virtual Machine. There is no apparent reason why the skilled person would modify Koizumi to introduce an extra intermediate representation. Similarly, if the teaching of Koizumi was applied to Java source code directly, a different intermediate representation would result,

one derived directly from the source code rather than the bytecode. Accordingly, claim 19 would also not be obvious in light of Koizumi.

Thus, reconsideration and withdrawal of the rejection of the claims are respectfully requested.

CONCLUSION

Based on the foregoing remarks, the Applicants respectfully request reconsideration and withdrawal of the rejection of claims and allowance of this application. To facilitate prosecution, the Applicants would appreciate an opportunity to discuss this matter with the Examiner in an interview.

AUTHORIZATION

The Commissioner is hereby authorized to charge any additional fees which may be required for consideration of this Amendment to Deposit Account No. 13-4500, Order No. 4362-4002.

In the event that an extension of time is required, or which may be required in addition to that requested in a petition for an extension of time, the Commissioner is requested to grant a petition for that extension of time which is required to make this response timely and is hereby authorized to charge any fee for such an extension of time or credit any overpayment for an extension of time to Deposit Account No. 13-4500, Order No. 4362-4002.

Respectfully submitted,
MORGAN & FINNEGAN, L.L.P.

Dated: 12/21/06 By: _____

James Hwa
Registration No. 42,680
(202) 857-7887 Telephone
(202) 857-7929 Facsimile

Correspondence Address:
MORGAN & FINNEGAN, L.L.P.
3 World Financial Center
New York, NY 10281-2101